

Haskell CheatSheet

Laborator 6

Tipuri de bază

```
5      :: Int
'H'    :: Char
"Hello" :: String
True   :: Bool
False  :: Bool
```

Determinarea tipului unei expresii

```
:t
> :t 42
42 :: Num a => a

a reprezintă o variabilă de tip, restrictionată la toate tipurile numerice.

> :t 42.0
42 :: Fractional a => a
```

În acest exemplu, **a** este restricționată la toate tipurile numerice fracționare (e.g. **Float**, **Double**).

Constructorii liste

```
[]      (:)
[]      -- lista vida
(:)     -- operatorul de adaugare
-- la inceputul listei

1 : 3 : 5 : [] -- lista care contine 1, 3, 5
[1, 3, 5]    -- sintaxa echivalenta
```

Operatorii logici

```
not && ||

not True      False
not False     True
True || False True
True && False  False
```

Operatorii pe liste

(++) head tail last init take drop

```
[1, 2] ++ [3, 4]      [1, 2, 3, 4]

head [1, 2, 3, 4]     1
tail [1, 2, 3, 4]    [2, 3, 4]

last [1, 2, 3, 4]    4
init [1, 2, 3, 4]   [1, 2, 3]

take 2 [1, 2, 3, 4]  [1, 2]
take 2 "HelloWorld" "He"

drop 2 [1, 2, 3, 4]  [3, 4]

null []              True
null [1, 2, 3]      False
```

Alte operații

length elem reverse

```
length [1, 2, 3, 4]      4

elem 3 [1, 2, 3, 4]     True
elem 5 [1, 2, 3, 4]     False

reverse [1, 2, 3, 4]    [4, 3, 2, 1]
```

Tupluri

Spre deosebire de liste, tuplurile au un număr fix de elemente, iar acestea pot avea tipuri diferite.

```
import Data.Tuple

("Hello", True) :: (String, Bool)
(1, 2, 3)       :: (Integer, Integer, Integer)

fst ("Hello", True) "Hello"
snd ("Hello", True) True
swap ("Hello", True) (True, "Hello")
```

Funcții anonime (lambda)

\arg1 arg2 → corp

```
\x -> x          funția identitate
(\x y -> x + y) 1 2      3
let f = \x y -> x + y    legare la un nume
(f 1 2)           3
```

Definire funcții

```
-- if .. then .. else
factorial x =
    if x < 1 then 1 else x * factorial (x - 1)

-- guards
factorial x
    | x < 1 = 1
    | otherwise = x * factorial (x - 1)

-- case .. of
factorial x = case x < 1 of
    True -> 1
    _    -> x * factorial (x - 1)

-- pattern matching
factorial 0 = 1
factorial x = x * factorial (x - 1)
```

Curry

În Haskell funcțiile sunt, by default, în forma curry.

```
:t (+)
(+) :: Num a => a -> a -> a

:t (+ 1)
(+ 1) :: Num a => a -> a
```

Funcționale uzuale

map filter foldl foldr zip zipWith

```
map      :: (a -> b) -> [a] -> [b]
filter  :: (a -> Bool) -> [a] -> [a]
foldl   :: (a -> b -> a) -> a -> [b] -> a
zip      :: [a] -> [b] -> [(a, b)]
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]

map (+ 2) [1, 2, 3]      [3, 4, 5]

filter odd [1, 2, 3, 4]  [1, 3]

foldl (+) 0 [1, 2, 3, 4] 10
foldl (-) 0 [1, 2]       -3 (0 - 1) - 2
foldr (-) 0 [1, 2]       -1 1 - (2 - 0)

zip [1, 2] [3, 4]        [(1, 3), (2, 4)]
zipWith (+) [1, 2] [3, 4] [4, 6]
```